**Lab 9**

**Inheritance**

# Objectives

**Following programming skills will be acquired in this lab:**

* To understand class inheritance.
* To understand the use of **protected** class members.
* To practice implementation of a base and derived class.
* To understand multiple inheritance.

**Inheritance between Classes**

A key feature of C++ classes is inheritance. Inheritance allows to create classes which are derived from other classes, so that they automatically include some of its "parent's" members, plus its own.

|  |
| --- |
| // The class Animal contains information and functions related to all animals  class Animal  {  private:  int legs;  int arms;  int age;  public:  Animal();  ~Animal();  void eat();  void sleep();  void drink();  };  //each of the following operations is unique to your furry animals  class Cat : public Animal  {  public:  int furcolor;  void purr();  void fish();  void markTerritory();  }; |

Classes that are derived from others inherit all the accessible members of the base class. That means that if a base class includes a member A and we derive it to another class with another member called B, the derived class will contain both members A and B. In order to derive a class from another, we use a colon (:) in the declaration of the derived class using the following format:

## *class derived\_class\_name: public base\_class\_name { /\*...\*/ };*

The public access specifier may be replaced by any one of the other access specifiers protected and private. This access specifier limits the most accessible level for the members inherited from the base class: The members with a more accessible level are inherited with this level instead, while the members with an equal or more restrictive access level keep their restrictive level in the derived class.We can summarize the different access types according to who can access them in the following way:

|  |  |  |  |
| --- | --- | --- | --- |
| **Member function Access from different classes** | **Parent class Data members access specifiers** | | |
| **public** | **protected** | **private** |
| Accessibility from functions of parent class | yes | yes | yes |
| Accessibility from functions of derived classes | yes | yes | **no** |
| Accessibility from external functions | yes | **no** | **no** |

Where "not members" represent any access from outside the class, such as from main(), from another

As an example, suppose that we want to declare a series of classes that describe polygons like our CRectangle, or like CTriangle. They have certain common properties, such as both can be described by Diagram

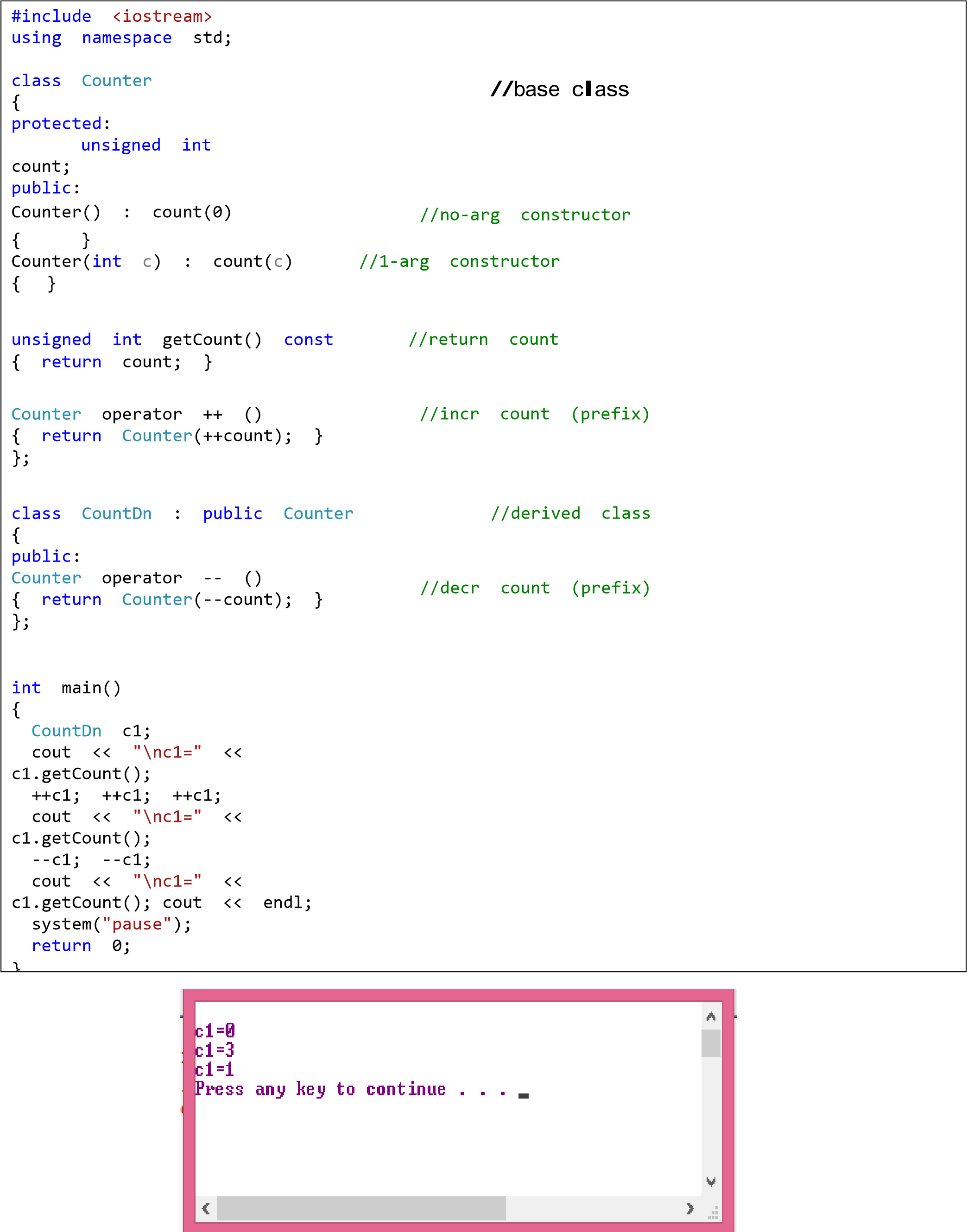
Description automatically generatedmeans of only two sides: height and base. This could be represented in the world of classes with a class Polygon from which we would derive the two other ones: Rectangle and Triangle.

The class Polygon would contain members that are common for both types of polygons. In our case: width and height. And Rectangle and Triangle would be its derived classes, with specific features that are different from one type of polygon to the other.

# Example 9.1

|  |
| --- |
| //Example 9.1  #include <iostream>  #include <string>  #include <conio.h>  using namespace std;  class Polygon  {  protected: int width, height;  public:  void setValues(int a, int b)  {  width = a;  height = b;  }  };  class Rectangle: public Polygon  {  public:  int area()  { return (width \* height); }  };  class Triangle: public Polygon  {  public:  int area()  { return (width \* height / 2); }  };  int main()  {  Rectangle rect;  Triangle trgl;    rect.setValues(4, 5);  trgl.setValues(4, 5);  cout << rect.area() << endl;  cout << trgl.area() << endl;    return 0;  } |

**Example 9.2**



**Leave It for Today’s Lab (This page only)**

**What is inherited from the base class?**

In principle, a derived class inherits every member of a base class except:

* its constructor and its destructor
* its operator=() members
* its friends

Although the constructors and destructors of the base class are not inherited themselves, its default constructor (i.e., its constructor with no parameters) and its destructor are always called when a new object of a derived class is created or destroyed.

If the base class has no default constructor or you want that an overloaded constructor is called when a new derived object is created, you can specify it in each constructor definition of the derived class:

***derived\_constructor\_name (parameters) : base\_constructor\_name (parameters) {...}***

# Example 9.3

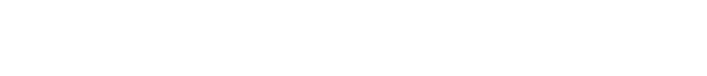
|  |  |
| --- | --- |
| // constructors and derived classes  #include <iostream>  using namespace std;  class mother  {  public:  mother()  { cout << "mother: no parameters\n"; }  mother(int a)  { cout << "mother: int parameter\n"; }  };  class daughter: public mother  {  public:  daughter(int a)  { cout << "daughter: int parameter\n\n"; }  };  class son : public mother  {  public:  son(int a) : mother(a)  { cout << "son: int parameter\n\n"; }  }; | int main()  {  daughter cynthia(0);  son daniel(0);  return 0;  }  **Output:**    **mother: no parameters daughter: int parameter**    **mother: int parameter son: int parameter** |

# 

# Multiple Inheritance

In C++ it is perfectly possible that a class inherits members from more than one class. This is done by simply separating the different base classes with commas in the derived class declaration. For example, if we had a specific class to print on screen (COutput) and we wanted our classes CRectangle and CTriangle to also inherit its members in addition to those of CPolygon we could write:

class Rectangle: public Polygon, public Output;



class Triangle: public Polygon, public Output;

## Example 9.4 Multiple Inheritance

|  |  |
| --- | --- |
| //Example 9.4  #include <iostream>  using namespace std;  class Polygon  {  protected:  int width, height;  public:  void setValues(int a, int b)  {  width = a;  height = b;  }  };  class Output  {  public:  void output(int i);  };  void Output::output(int i)  {  cout << i << endl;  }  class Rectangle : public Polygon, public Output  {  public:  int area() { return (width \* height); }  };  class Triangle : public Polygon, public Output  {  public:  int area() { return (width \* height / 2); }  }; | int main()  {  Rectangle rect;  Triangle trgl;  rect.setValues(4, 5);  trgl.setValues(4, 5);  rect.output(rect.area());  trgl.output(trgl.area());  return 0;  }  **Output**:    20  10 |

## Example 9.5: Multiple Inheritence

|  |
| --- |
| #include <iostream>  using namespace std;  const int LEN = 80;    class student  {  private:  char school[LEN];  char degree[LEN];  public:  void getedu()  {  cout << "Enter name of school or university: ";  cin >> school;  cout << "Enter highest degree earned \n";  cout << "(Highschool, Bachelor’s, Master’s, PhD): ";  cin >> degree;  }  void putedu() const  {  cout <<"\n School or university: "<< school;  cout <<"\n Highest degree earned: "<< degree;  }  }; |

|  |
| --- |
| class employee  {  private:  char name[LEN];  unsigned long number;  public:  void getdata()  {  cout << "\n Enter last name: ";  cin >> name;  cout << "\n Enter number: ";  cin >> number;  }    void putdata() const  {  cout << "\n Name: "<< name;  cout << "\n Number: "<< number;  }  }; |

|  |
| --- |
| class scientist: private employee, private student  {  private:  int pubs;  public:  void getdata() **//Concept of Function Overriding. As function with same name and signature in parent class as well as in derive class. So derive class fucntion will execute upon call from main().**  {  employee::getdata();  cout << " Enter number of publications: ";  cin >> pubs;  student::getedu();  }  void putdata() const  {  employee::putdata();  cout << "\nNumber of publications: "<< pubs;  student::putedu();  }  }; |

|  |
| --- |
| class manager: private employee, private student //management  {  private:  char title[LEN];  double dues;  public:  void getdata()  {  employee::getdata();  cout << " Enter title: ";  cin >> title;  cout << " Enter golf club dues: ";  cin>> dues;  student::getedu();  }  void putdata() const  {  employee::putdata();  cout << "\n Title: " <<title;  cout << "\n Golf club dues: " << dues;  student::putedu();  }  }; |

**Lab 9 Practice Exercises**

### Exercise 9.1

Write a program that defines a shape base class with a constructor that gives value to width and height. Then define two derived classes triangle and rectangle, that calculate the area of the shape. In the main, define two variables a triangle and a rectangle and then compute their area.

Code:

#include <iostream>

using namespace std;

class Shape

{

protected:

int width;

int height;

public:

void setValues(int a,int b)

{

set\_height(a);

set\_width(b);

}

int width1() const

{

return width;

}

void set\_width(int width)

{

this->width = width;

}

int height1() const

{

return height;

}

void set\_height(int height)

{

this->height = height;

}

};

class Rectangle : public Shape

{

public:

int area() { return(width \* height); }

};

class Triangle : public Shape

{

public:

int area() { return (width \* height / 2); }

};

int main()

{

Rectangle recta;

recta.setValues(5, 6);

Triangle tri;

tri.setValues(10, 12);

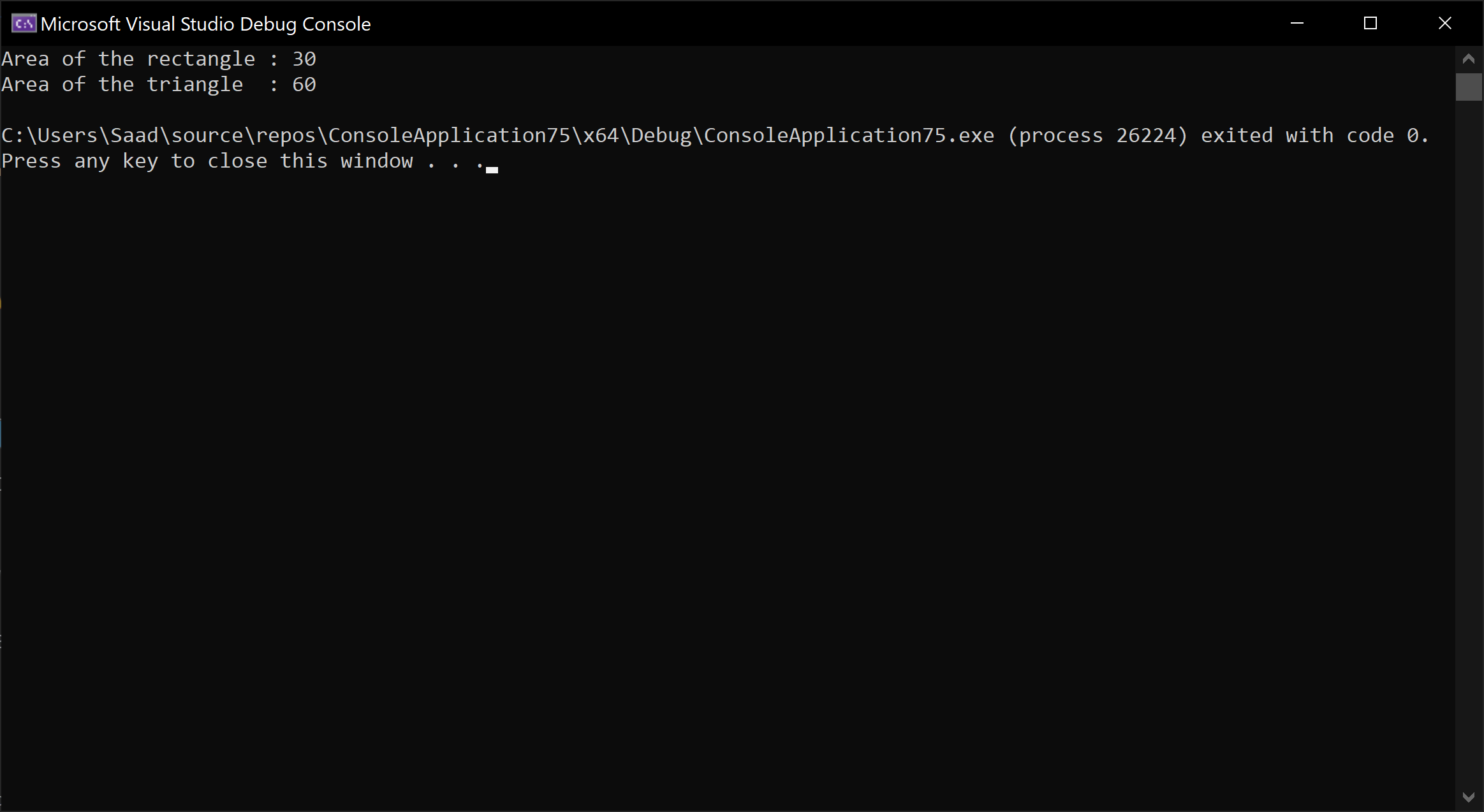
cout << "Area of the rectangle : " << recta.area() << endl;

cout << "Area of the triangle : " << tri.area() << endl;

return 0;

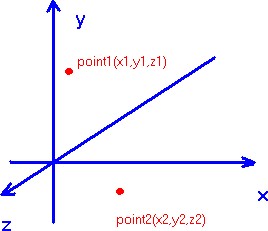
}

Output:



### Exercise 9.2

Write a class to represent a 2D point. Derive a class from it to represent a 3D point. Use objects of the derived classes to find distance between two points as shown below:



Code:

#include <iostream>

using namespace std;

class twoD\_Point{

protected:

int x;

int y;

public:

void set\_x(int x)

{

this->x = x;

}

void set\_y(int y)

{

this->y = y;

}

int get\_x() const

{

return x;

}

int get\_y() const

{

return y;

}

};

class threeD\_Point : public twoD\_Point

{

private:

int z;

public:

void setValue(int a, int b, int c)

{

set\_x(a);

set\_y(b);

set\_z(c);

}

void set\_z(int z)

{

this->z = z;

}

int get\_z() const

{

return z;

}

};

int main()

{

threeD\_Point t1,t2;

t1.setValue(1, 2, 3);

t2.setValue(2, 5, 8);

float distance;

int s;

s = pow((t2.get\_x() - t1.get\_x()), 2)+

pow((t2.get\_y() - t1.get\_y()), 2)+

pow((t2.get\_z() - t1.get\_z()), 2);

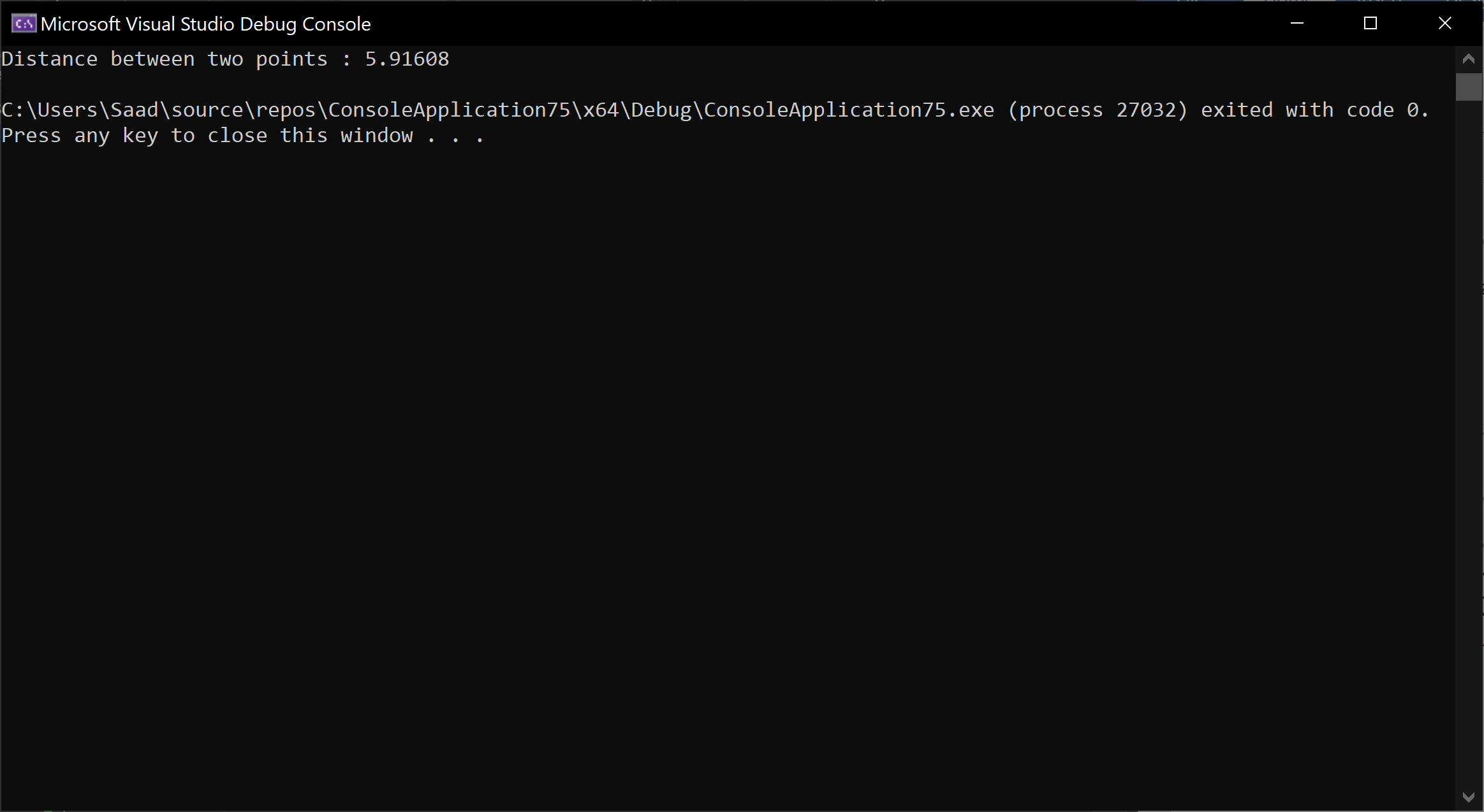
distance = sqrt(s);

cout << "Distance between two points : " << distance << endl;

return 0;

}

Output:



### Exercise 9.3

Create a class called **Employee** whose objects are records for an employee. This class will be a derived class of the base class [**Person**. A](http://www.cs.bham.ac.uk/~mdr/teaching/RedHotChilli/java/Person.java)n employee record has an employee's name (inherited from the class Person), an annual salary represented as a single value of type double, hiring year and a national insurance number, which is a value of type String.

Your class should have a reasonable number of constructors and accessor functions, as well as a print function. Write a driver to fully test your class definition for 10 employees.

Code:

#include <iostream>

using namespace std;

class Person

{

protected:

string name;

public:

string get\_name() const

{

return name;

}

void set\_name(string name)

{

this->name = name;

}

};

class Employee : public Person

{

private:

double salary;

int hiring\_year;

string nin;

public:

Employee(string n = "" ,double a = 0, int b = 0, string c = "")

{

set\_name(n);

set\_salary(a);

set\_hiring\_year(b);

set\_nin(c);

}

void Input()

{

string n ;

cout << "Enter name :" << endl;

cin >> n;

set\_name(n);

cout << "Enter Salary : " << endl;

cin >> salary;

cout << "Enter Hiring Year : " << endl;

cin >> hiring\_year;

cout << "Enter National Insurance Number : " << endl;

cin >> nin;

}

double get\_salary() const

{

return salary;

}

void set\_salary(double salary)

{

this->salary = salary;

}

int get\_hiring\_year() const

{

return hiring\_year;

}

void set\_hiring\_year(int hiring\_year)

{

this->hiring\_year = hiring\_year;

}

string get\_nin() const

{

return nin;

}

void set\_nin(string nin)

{

this->nin = nin;

}

void print()const

{

cout << "Employee name : " << get\_name() << endl;

cout << "Annual Salary : " << get\_salary() << endl;

cout << "Hiring Year : " << get\_hiring\_year() << endl;

cout << "National Insurance Number : " << get\_nin() << endl;

}

};

int main()

{

Employee e[10];

for(int i = 0; i < 10; i++)

{

e[i].Input();

}

for(int i =0; i < 10; i++)

{

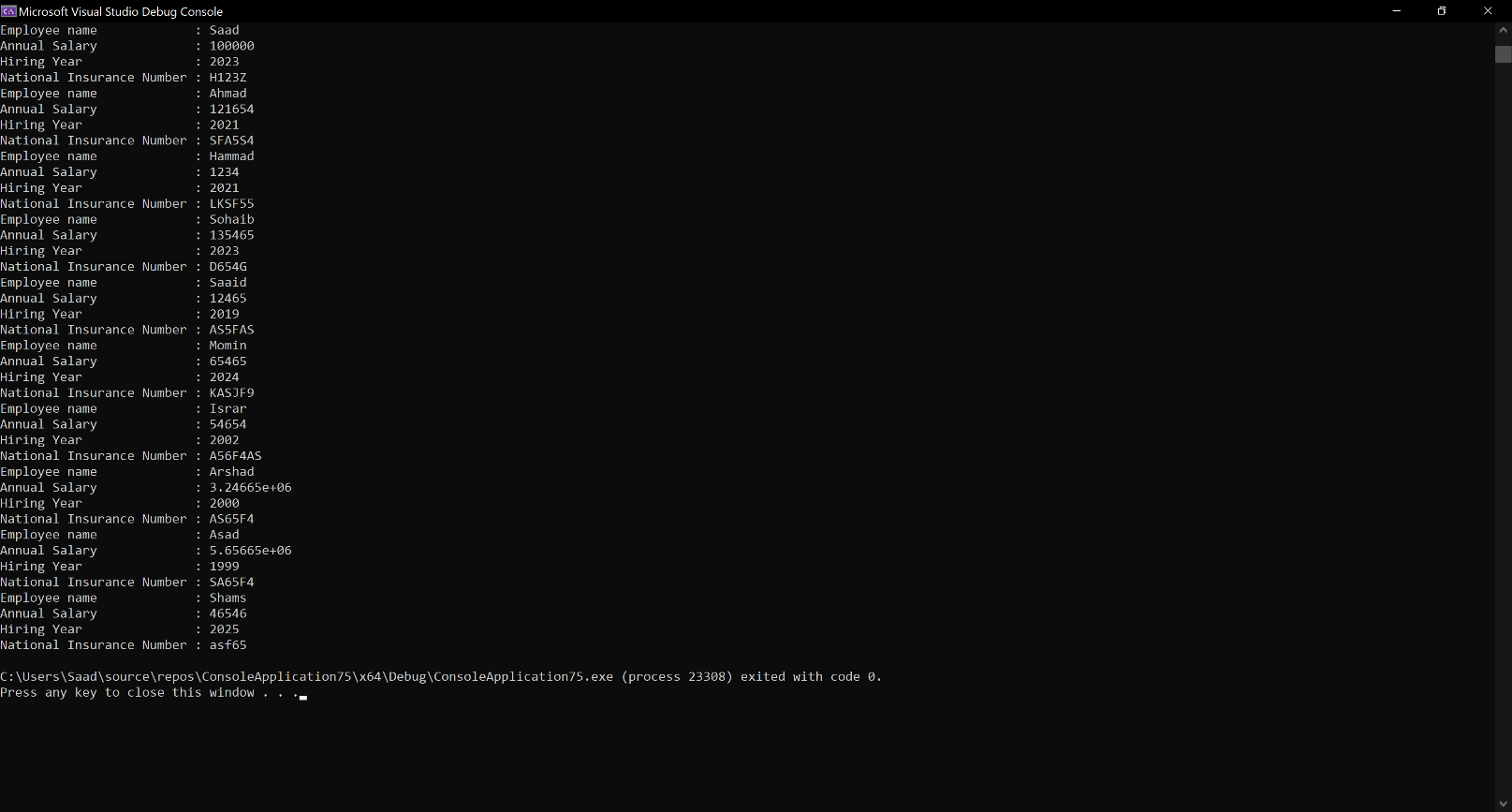
e[i].print();

}

return 0;

}

Output:

****

**Leave it for now (9.4)**

**Optional Exercise 9.4**

The examination department of an institute is interested in automating the schedule of examination. It is required to store the Date, Time, duration, and course title for each exam. Two types of exams are conducted by the department, theory and practical. For theory exams, the invigilation is carried out by the respective teacher and the name of invigilator is to be stored in the database. All theory exams are conducted in a dedicated examination hall. The practical exams are conducted in the labs and for each practical exam the location (Lab number) of the exam needs to be stored. The lab attendant of the lab conducts the exam. At the end of exam, each invigilator is paid an amount. For theory exams the amount is 20$ per hour while for lab exams it is 10$ per hour.

Design an object-oriented solution to the above problem and implement your solution in C++.